**数据库设计规范**

**目录**

[**一、** **命名规范** 2](#_Toc486493097)

[**1.** **子系统命名** 3](#_Toc486493098)

[**2.** **表名命名** 3](#_Toc486493099)

[**3.** **列名命名** 3](#_Toc486493100)

[**4.** **索引命名** 3](#_Toc486493101)

[**5.** **主键、外键命名** 4](#_Toc486493102)

[**6.** **整体样例** 4](#_Toc486493103)

[**二、** **索引规范** 6](#_Toc486493104)

[**1.** **索引设计原则** 6](#_Toc486493105)

[**2.** **索引字段选择** 6](#_Toc486493106)

[**3.** **复合索引字段选择** 6](#_Toc486493107)

[**三、** **设计规范** 7](#_Toc486493108)

[**1.** **数据表主键统一使用UUID** 7](#_Toc486493109)

[**2.** **表中应该避免可为空的列** 7](#_Toc486493110)

[**3.** **表不应该有重复的值或者列** 8](#_Toc486493111)

[**4.** **表中记录应该有一个唯一的标识符** 8](#_Toc486493112)

[**5.** **数据库对象要有统一的前缀名** 8](#_Toc486493113)

[**6.** **尽量只存储单一实体类型的数据** 8](#_Toc486493114)

[**7.** **必须符合数据库的第一范式** 8](#_Toc486493115)

[**8.** **字符串类型字段尽量选择varchar而不用nvarchar** 8](#_Toc486493116)

[**9.** **除核心表需要创建数据库级外键关联，其他表建议使用程序级逻辑外键关联，两种外键字段应创建索引** 8](#_Toc486493117)

[**10.** **字符集选择UTF-8以兼容多国字符和系统迁移兼容性** 8](#_Toc486493118)

[**11.** **避免使用触发器** 8](#_Toc486493119)

[**12.** **增加数据表的审计功能** 8](#_Toc486493120)

[**四、** **SQL规范** 9](#_Toc486493121)

[**1.** **SQL代码规范** 9](#_Toc486493122)

[**2.** **SQL性能规范** 10](#_Toc486493123)

[**1)** **SQL简化和减少数据库查询次数，建议关联表数量不要超过3个** 10](#_Toc486493124)

[**2)** **在查询列上避免使用表达式** 10](#_Toc486493125)

[**3)** **查询条件列类型的隐含转换** 11](#_Toc486493126)

[**4)** **Like查询** 11](#_Toc486493127)

[**5)** **排序，分组查询** 13](#_Toc486493128)

[**6)** **视图，存储过程，函数，触发器的使用** 13](#_Toc486493129)

[**7)** **使用批操作提高数据插入效率** 14](#_Toc486493130)

[**8)** **慎用union或union all** 14](#_Toc486493131)

[**9)** **优化嵌套查询** 14](#_Toc486493132)

1. **命名规范**

表名，字段名，视图名，存储过程名，函数名，触发器名称统一使用大写字母命名。使用最接近数据库对象含义的英文单词，单词组合，或缩写来命名，尽可能做得见名知意。

* 1. **子系统命名**

用户管理-SY

佣金管理-YJ

评价管理-EM

业务系统-BS

业务管理-BP

渠道业务-CB

公共管理-PG

* 1. **表名命名**

表名采用系统名的2位缩写，容易理解的表名英文单词一个或多个（建议不超过3个），以及有主表（M）历史表（H）区分时使用代号，各部分以下划线分隔，总长度建议不超过30位。如：资金方评价主表 EM\_FUND\_EVALUATE\_M

* 1. **列名命名**
* 字段名为有意义的单词，或单词的缩写
* 如果字段由几个单词组成，则单词间用下划线(“\_”)分割，如：资金方用户唯一标识 FUND\_USERID。
* 字段名建议在30个字符内。当字段名超过30字符时，可用缩写来减少字段名的长度。
  1. **索引命名**
* 索引须按照IDX\_table\_<column>,其中<table>是建立索引的表名，<column>是建立索引的字段名，如果字段名为多个单词则使用缩写。
* 索引名限制在30个字符内。当索引名超过30字符时，可用缩写来减少索引名的长度，如description –> desc；information –> info；address –> addr等。
  1. **主键、外键命名**
* 主键按照PK\_<table>的规则命名，其中<table>为数据库表名
* 唯一键按照UK\_<table>\_<column>的规则命名，其中<table>为数据库表名，<column>为字段名
* 外键按照FK\_<ftable>\_<ptable>\_<column>的规则命名，其中<ftable>为外键表表名，<ptable>为主键表表名，<column>为外键字段名
  1. **整体样例**

其他未详细说明的对象，命名规范可按前缀+功能描述英文单词的方式进行设计，具体样例参考下表：

|  |  |  |
| --- | --- | --- |
| **对象名** | **前缀** | **范例** |
| **表（table）** | 子系统缩写 | **SY\_USER\_INFO** |
| **视图（view）** | V\_ | **V\_USER\_INFO** |
| **序列（sequence）** | SEQ\_ | **SEQ\_USER\_INFO** |
| **簇（cluster）** | C\_ | **C\_USER\_INFO** |
| **触发器（trigger）** | TRG\_ | **TRG\_USER\_INFO** |
| **存储过程（procedure）** | SP\_ | **SP\_USER\_INFO** |
| **函数（function）** | FN\_ | **FN\_USER\_INFO** |
| **物化视图（materialized view）** | MV\_ | **MV\_USER\_INFO** |
| **包和包体（package & package body）** | PKG\_ | **PKG\_USER\_INFO** |
| **类和类体（type & type body）** | TYP\_ | **TYP\_USER\_INFO** |
| **主键（primary key）** | PK\_ | **PK\_USER\_INFO** |
| **外键（foreign key）** | FK\_ | **FK\_USERINFO\_USERBASIC\_FIELDNAME** |
| **唯一索引（unique index）** | UK\_ | **UK\_USER\_INFO\_FIELDNAME** |
| **普通索引（normal index）** | IDX\_ | **IDX\_USER\_INFO\_FIELDNAME** |
| **位图索引（bitmap index）** | BK\_ | **BK\_USER\_INFO\_FIELDNAME** |
| **同义词（synonym）** | 依据所分配的表所属模块/模式 |  |
| **数据库链接（database link）** | 无特殊要求 |  |

1. **索引规范**
   1. **索引设计原则**

* 原则上表索引的个数不能超过5个；
* 原则上单个字段上的索引不能超过2个；
* 原则上复合索引引用的字段不能超过3个字段；
* 原则上分区表的索引类型全部使用LOCAL索引；
* 配置数据类的表，如数据量比较少，如少于300条记录，除了主键外原则上不建索引
* 频繁做更新操作的表，尽可能减少索引数量或者不建索引，保证效率，如流水表
* 索引引用字段的顺序尽可能与使用该索引的查询中ORDER BY字段顺序保持一致
  1. **索引字段选择**
* 频繁出现在where字句里的字段；
* 频繁用来和其他表关联的字段；
* 有高的选择性和过滤性的字段；
* 尽量不要在较长字符串的字段上建立索引，如varchar2(1000);
* 建立索引的时候，建议考虑到select和insert,update,delete的平衡；
* 一般建议在查询数据量**10%**以下使用索引
  1. **复合索引字段选择**
* 如果所有字段查询频率相同，则把**选择性好**的字段放在前面；
* 复合索引的几个字段是否经常同时以AND方式出现在Where子句中？单字段查询是否极少甚至没有？如果是，则可以建立复合索引；否则考虑单字段索引；
* 如果复合索引中包含的字段经常单独出现在Where子句中，则分解为多个单字段索引；
* 如果复合索引所包含的字段超过3个，那么仔细考虑其必要性，考虑减少复合的字段；
* 如果既有单字段索引，又有这几个字段上的复合索引，一般可以删除复合索引
* where子句的查询条件构成索引字段前沿列，**频繁**查询的字段放在前面；
* 如果所有字段查询频率相同，则把**排列顺序**的字段放在前面；
* 尽量保证复合索引中至少有一个字段具有非空约束;
* 复合索引使用规律:索引index(a,b,c) ，在where 条件里，(a)、(a,b)、（a,b,c）、（a,c）组合可用到索引;(b)、（b,c）、（c）组合用不到索引。

1. **设计规范**
   1. **数据表主键统一使用UUID**
   2. **表中应该避免可为空的列**

虽然表中允许空列，但是空字段是一种比较特殊的数据类型，数据库在处理的时候，需要进行特殊的处理。如此的话就会增加数据库处理记录的复杂性，当表中有比较多的空字段时，在同等条件下，数据库处理的性能会降低许多。

建议方式：

* 通过设置默认值的形式，来避免空字段的产生

字符型的默认值为一个空字符值串’’；数值型的默认值为数值0；逻辑型的默认值为数值0，其中：系统中所有逻辑型中数值0表示为“假”，数值1表示为“真”；日期型/日期戳型默认为‘1970-1-2‘。

* 创建副表保存空字段

如果一张表中，允许为空的列比较多，接近表全部列数的三分之一，而且这些列在大部分情况下都是可有可无的，建议另外建立一张副表，以保存这些列。

* 1. **表不应该有重复的值或者列**
  2. **表中记录应该有一个唯一的标识符**
  3. **数据库对象要有统一的前缀名**
  4. **尽量只存储单一实体类型的数据**
  5. **必须符合数据库的第一范式**

尽量达到第二范式及第三范式，从实际情况出发，取平衡点，即不允许字段出现二义性。

* 1. **字符串类型字段尽量选择varchar而不用nvarchar**
  2. **除核心表需要创建数据库级外键关联，其他表建议使用程序级逻辑外键关联，两种外键字段应创建索引**
  3. **字符集选择****UTF-8以兼容多国字符和系统迁移兼容性**
  4. **避免使用触发器**
  5. **增加数据表的审计功能**

表中增加如下四个字段实现审计：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 描述 | 类型 | 说明 |
| OPERATE\_USER | 操作用户 | Varchar(32) |  |
| OPERATE\_TIME | 操作时间 | TIME |  |
| STATUS | 记录状态 | Char(1) | 0-无效 1-有效 |
| VERSION | 记录版本 | int | 1. 可以保证并发操作。更新数据时将版本号带入条件，可以在无需数据库锁介入的情况实现并发操作，在分布式环境比较适用。 2. 可以记录数据历史，在没有历史表的情况下可以记录数据历史。 3. 可以跟踪ddl变更历史，当ddl更新时维护ddl版本表，dml操作时获取ddl版本作为记录版本。   三种方式可选任意功能作为用途。 |

1. **SQL规范**
   1. **SQL代码规范**

* 所有的关键字的所有字母必须大写。
* 表名、列名、视图名或它们的别名必须和它们的定义保持一致。
* 注释必须要规范。
* 关联查询禁止使用“SELECT Ci.\* FROM customerinfo ci, salesinfo sf WHERE ci. CustomerID=sf. CustomerID” 写法，换成inner join关键字的写法以增强可读性。
  1. **SQL性能规范**
     1. **SQL简化和减少数据库查询次数，建议关联表数量不要超过3个**

在满足业务需求的前提下，尽可能使编写的SQL简单，不要选择多余的字段和不必要的嵌套查询，越简单的SQL，查询性能可能越好，也越好进行性能优化。

比如：

不要用SELECT \*：SELECT语句中写出必要的要选择的全部列名，增强语句可读性，避免不必要的选择；SELECT \*增加了对所有字段的依赖，当表增加了字段后，有可能发生错误；此外还可能增加了数据的流量，查询了一些实际不需要的字段。

其它SQL查询也要求选择真正需要选择的字段，避免选择出多余的字段。

另外，尽可能减少数据库操作的次数，建议一次性查出你需要的结果集，先放到应用内存中进行遍历处理。

如果经常查询的关联语句中关联表数量超过3个，建议在表设计时在表中冗余关联表主键，已减少关联表数量。

* + 1. **在查询列上避免使用表达式**

在WHERE中，数据库函数、计算表达式等等，要尽可能将放在等号右边。否则会使所比较的字段上的索引失效；

SELECT　＊

FROM　service\_promotion

WHERE TO\_CHAR(gmt\_modified,’yyyy-mm-dd’)

= ‘20001-09-01’;

而应使用：

SELECT \*

FROM service\_promotion

WHERE gmt\_modified

>= DATE\_FORMAT('2009-07-26 20:49:33','%Y-%m-%d');

AND gmt\_modified

< DATE\_FORMAT('2009-07-26 20:49:33','%Y-%m-%d');

* + 1. **查询条件列类型的隐含转换**

尽量注意比较值与查询列数据类型的一致性(int与int比较、char与char比较)，避免使用数据库的类型自动转换功能,比如：

SELECT \* FROM category

WHERE id = ‘123’;

-- id在表中定义为int类型

* + 1. **Like查询**

在进行模糊查询时，建议不要使用WHERE columnname like ‘%字符串%’这样的查询形式，这样的查询语句不能使用列上所建的索引，当数据量稍微有点大就会导致有严重的性能问题。如果确实有这样的需求，应该考虑用别的方式实现，这个问题需要在表设计的时候就考虑到。

模糊查询分为三种，前模糊（‘%abc’）、后模糊（‘abc%’）、全模糊（‘%abc%’），对三种模糊查询方式的建议替换方式：

* 方式一

前模糊（‘% abc’）:

前面有%号，通过create index idx\_xx on tab(reverse(xx)),然后sql 可改写为

   改写前:

   select \* from test where owner like '%abc'

   改写后：

   select \* from test where reverse(owner) like reverse('%abc');

后模糊（‘abc%’）:

Mysql对后模糊查询优化可以走索引。

全模糊（‘%abc%’）:

创建新表保存原表的like列和rowid，然后通过rowid关联查询，然后sql改写如下：

   改写前：

   select \* from test where owner like '%SCOTT%';

   改写后：

   select 原表.\* from 新表 t1 join 原表 t2 on t1.rid = t2.rowid where 新表.owner like '%SCOTT%';

* 方式二

多字段like模糊查询优化：

最常见的写法：where  a like '%xx%' or b like '%xx%' or c like '%xx%';  这种写法查询效率低，经过调查，下面的方法可以替代，并且效率高：

1、如果like的关键字相同：

where instr(nvl(a, '')||nvl(b,'')||nvl(c,''),   'xx') > 0

把要模糊查询的字段先拼接起来，拼接时需要把null转成‘’，否则只要有一个字段值是空，整个拼接的字符串都成空了， 然后用instr 函数去过滤；

2、如果like的关键字不同：

where instr(a, 'xx') > 0  or  instr(b, 'yy') > 0  or instr(c, 'zz') > 0

经过测试，这两种方法都比like效率要高；

* 方式三

使用locate替换

table.field like  ‘%AAA%’ 可以改为 locate (‘AAA’ , table.field) > 0

其他情况应视具体场景进行优化，尽量避免使用模糊查询。

* + 1. **排序，分组查询**

大量的排序操作将严重影响系统性能，所以尽量减少order by和group by排序操作。如果有大数据量的表确实存在大范围的分组查询求和运算，建议通过建立临时统计表，通过触发器或后台作业来完成统计数据的计算。对大数据量表的排序查询，除可以在排序列上建立索引外，应该尽量通过表设计加条件减少排序范围来实现排序操作。

对于进行分组求和操作的SQL，可以通过增加选择项来避免排序，比如,在sql语句的末尾增加order by null选项避免进行资源消耗量极大的排序操作。

Select id,sum(moneys) from sales2 group by id order by null;

* + 1. **视图，存储过程，函数，触发器的使用**

复杂，重复性使用率高的SQL查询语句，建议定义成视图（VIEW）来使用，好处是将来可以在数据库后台进行修改，优化等维护，同时也可以简化应用端的代码编写。对操作任务量大，比如定时性的数据统计，计算任务，建议编写成存储过程，函数来实现。尽量少用触发器，特别是使用带有大量数据操作任务的触发器，那样的触发器会降低表的更新速度。而且大量使用触发器会给数据库的维护带来更大的难度。

* + 1. **使用批操作提高数据插入效率**

如果数据插入量比较大，建议写成批量操作的形式，将能大大提高数据库插入的效率，比如：

INSERT INTO `T\_VISITIP` VALUES

(38094,'10.10.10.0',10,'2007-02- 28'),

(38095,'10.10.10.1',10,'2007-02- 28'),

(38096,'10.10.10.2',10,'2007-02- 28'),

(38097,'10.10.10.3',10,'2007-02- 28') ,

………………………………..;

begin;

INSERT INTO a VALUES (1,23),(2,34),(4,33);

INSERT INTO a VALUES (8,26),(6,29);

………………………

commit;

* + 1. **慎用union或union all**

慎用union或union all，特别是合并后再进行排序操作的SQL,碰到数据量比较大时，进行优化会非常困难。

* + 1. **优化嵌套查询**

使用连接（JOIN）来代替子查询(Sub-Queries) ，在有的情况下可能能大大提高sql执行的速度。

子查询：

SELECT \* FROM customerinfo WHERE CustomerID in (SELECT CustomerID FROM salesinfo )

改写成：

SELECT Ci.\* FROM customerinfo ci, salesinfo sf

WHERE ci. CustomerID=sf. CustomerID